Journal of Computer and System Sciences 81 (2015) 671–691

Contents lists available at ScienceDirect

Journal of Computer and System Sciences

www.elsevier.com/locate/jcss

Optimizing large data transfers in parity-declustered data layouts

Eric J. Schwabe *

School of Computing, DePaul University, 243 S. Wabash Ave., Chicago, IL 60604, USA

Ian M. Sutherland

ARTICLE INFO

Article history:
Received 1 April 2011
Received in revised form 7 November 2013
Accepted 13 November 2014
Available online 27 November 2014

Keywords:
RAID
Disk arrays
Fault tolerance
Data layouts
Array codes

ABSTRACT

Disk arrays allow faster access to users' data by distributing the data among a collection of disks and allowing parallel access. Fault tolerance in a disk array can be achieved by using a data layout, and the technique of parity declustering allows faster failure recovery at the cost of additional space dedicated to redundant information. A collection of six performance conditions that parity-declustered data layouts should satisfy has guided most previous work; however two of these conditions (Maximal parallelism and Large write optimization) cannot be jointly satisfied in most cases. This limits the ability of parity-declustered data layouts to take full advantage of the available parallelism during large data transfers. We present data layouts that approximately satisfy these two conditions simultaneously for all possible array configurations, and bound the deviations from complete satisfaction. Our results yield improved performance guarantees for large data transfers in parity-declustered data layouts.

1. Introduction

1.1. Data layouts for disk arrays

Disk arrays provide increased transfer rates for large data sets by distributing the data over a collection of disks, rather than a single disk, and allowing parallel access. Since each disk in the array may fail with some probability in unit time, the probability that some disk in a large array will fail in unit time is greatly increased. Thus in order to be practical, large disk arrays must have the ability to reconstruct the contents of a failed disk.

Fault tolerance in a disk array can be achieved by constructing a *data layout*, which is an arrangement of the data along with some additional redundant information that allows the array to reconstruct the contents of one or more failed disks. A data layout is constructed by dividing each disk into the same number of equally sized *units* and partitioning the set of all units in the array into a collection of non-overlapping sets of equal size called *stripes*. The number of units on each disk is called the *depth* of the layout, and the number of units in each stripe is called the *stripe size*. Most of the units in each stripe hold users' data; however, one or more units in each stripe instead hold some encoded redundant information that is computed from the data stored in the other units of the stripe. This redundant information stored in each stripe enables the array to recover from disk failures by reconstructing the contents of one or more failed disks.

* Corresponding author. Tel.: +1 312 362 5943; Fax: +1 312 362 6116.
E-mail addresses: eschwabe@cdm.depaul.edu, sans_peur2001@yahoo.com.

http://dx.doi.org/10.1016/j.jcss.2014.11.013
0022-0000/

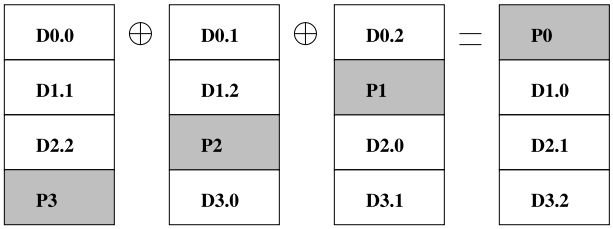

Fig. 1. RAID5 on a four-disk array. In each row **i**, the parity unit **Pi** is the bitwise exclusive "or" of the three data units **Di.0**, **Di.1**, and **Di.2**.

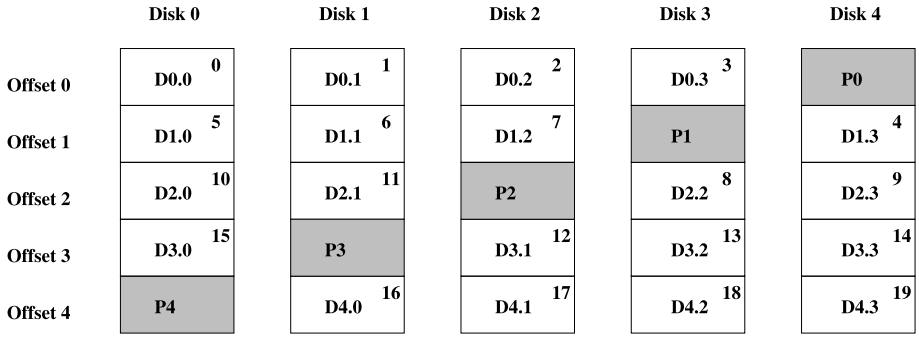

Fig. 2. RAID 5 on a five-disk array, illustrating disk numbers, offsets, and a mapped linear address space.

For example, in a RAID5 data layout (due to Patterson, Gibson, and Katz [15]), each stripe consists of one unit from each of the v disks. Some set of $v-1$ units in each stripe hold data (we call these the *data units*), and the remaining unit (which we call the *parity unit*) holds the bitwise exclusive "or" of the $v-1$ data units. Thus the stripe size is v, and the depth of the layout is also v, though it can be made to be any multiple of v by repeating the layout. (In Fig. 1, the array size, stripe size, and depth are all equal to four, but the illustrated layout can be repeated as many times as desired to construct a layout whose depth is any multiple of four.)

A v-disk array using a RAID5 data layout can recover from any single disk failure by reading the remaining contents of each of the surviving disks. In particular, the contents of each unit on the failed disk can be reconstructed by reading the $v-1$ remaining units in the stripe and taking their bitwise exclusive "or". Repeating this for each unit on the failed disk will require each unit on the remaining disks to be read exactly once. Thus RAID5 achieves single-fault tolerance at the cost of dedicating a $1/v$ fraction of the total space in the array to redundant information.

A data layout is a type of *array code*, which is an erasure-correcting code where the contents of the disk array are represented as a rectangular array of words that is divided into blocks. Redundant information for each block is computed using only bitwise exclusive-or operations on the data words in the block. In particular, data layouts are *systematic* array codes, in which the original data is stored in unencoded form somewhere in the disk array. Plank gives an overview of single- and multiple-fault-tolerant array codes [13], including EVENODD codes [5,6], Blaum–Roth codes [7], RDP codes [8,4], STAR codes [11], and Liberation codes [14].

Clients using a disk array to store their data need not be concerned with the details of the data layout. In particular, they need not know which units contain data or redundant information, or even on which disks and at which offsets their data are stored. Rather, a linear address space is mapped onto the set of data units of the data layout, so the data will appear to reside on a single logical disk using these addresses. (See Fig. 2 for an example of a RAID5 layout with a linear address space mapped onto its data units.) Throughout the paper, we will consider a data layout to include such a mapping of a linear address space to its data units.

If an array must remain available during the reconstruction of lost data, or must be taken off-line for as little time as possible for the purpose of failure recovery, it may be desirable to dedicate more space to redundant information if it allows failure recovery to be completed more quickly. This tradeoff of additional redundant space for reduced failure recovery time can be achieved using a technique called *parity declustering*, in which the stripe size k is chosen to be smaller than the array size v. Parity-declustered data layouts have been considered by, among others, Muntz and Lui [12], Holland and Gibson [10], Stockmeyer [20], Schwabe and Sutherland [16,18], Schwabe, Sutherland, and Holmer [19], Alvarez, Burkhard, and Cristian [1], and Alvarez, Burkhard, Stockmeyer, and Cristian [2].

A parity-declustered data layout is a single-fault-tolerant systematic array code in which more than one disk's worth of space is dedicated to redundant information. Most array codes that use more than one disk's worth of space for redundant information use that space to increase the level of fault tolerance in the disk array rather than to speed up the reconstruction of a single failed disk.

Holland and Gibson [10] described the following six conditions that data layouts (and their associated data mappings) should satisfy: